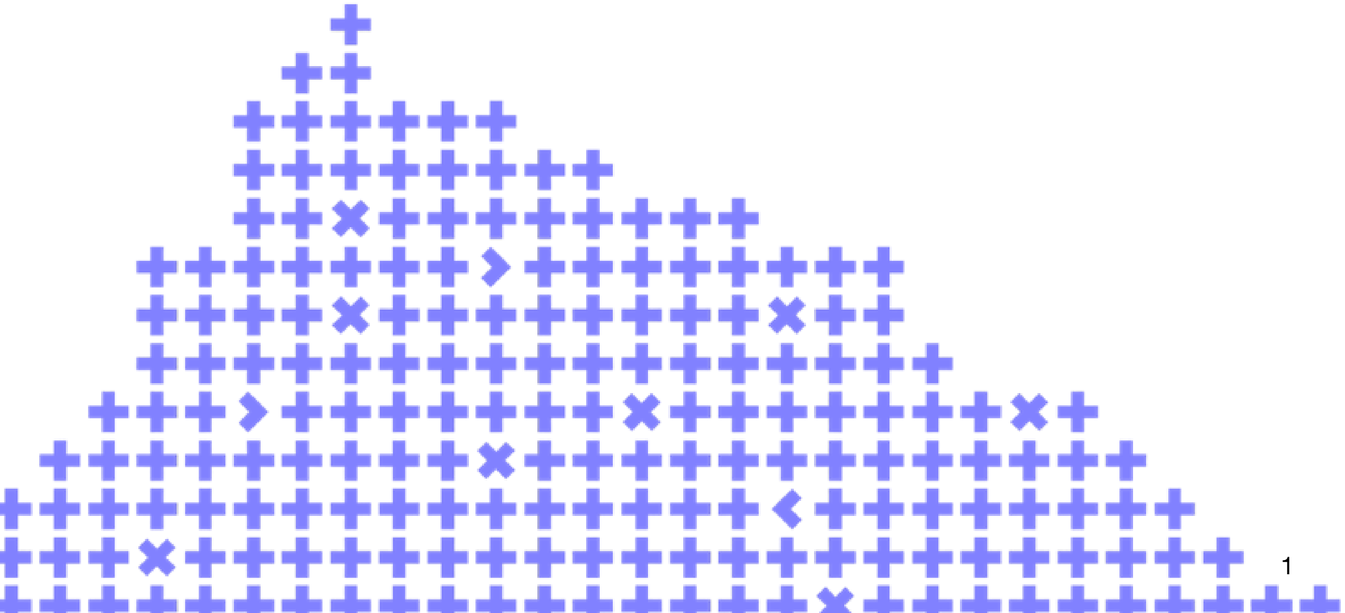
Everyday Practical Vectorization

Ignas Bagdonas

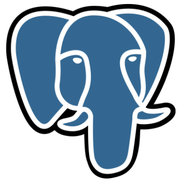


High Load++ Armenia

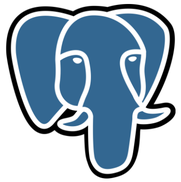
Co-organizer

Yandex

O(1) Shapes of Highload



O(log n) Shapes of Highload



O(n) Shapes of Highload

O(n * log n) Shapes of Highload

O(n²) Shapes of Highload

O(n!) Shapes of Highload

O(n!) Shapes of Highload

O(n!) Shapes of Highload

Who’s talking?

A network plumber with hardware and software clue.

- Network architecture and engineering.
- BGP, Verilog, C.
- More packets, less seconds.
- System-level design.
- Ex-Cisco, ex-Equinix.

- Fragments of production systems.

The Context

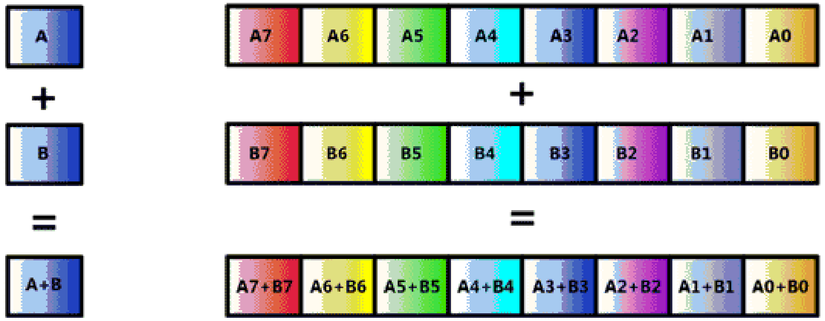
- Vector extensions have been present on x86 platforms in various forms for over a quarter of century.
- And have accumulated a substantial amount of misconceptions and myths.

A look from two sides:

- How one could use it in domains other than High-Performance Computing.
- How one could use it without a resulting loss of performance.

Vectorization?

- The logic of a scalar operation (and eventually an algorithm) applied to multiple sets of data.
- Not a novel concept as such, has been around for half a century.
- Predominantly a domain of HPC and specialized compute platforms.
- Performance characterization in terms of latency and throughput.
- SIMD as a specific instantiation of vectorization approach on x86 platforms.



Practical and Everyday: applicable to a multitude of generic and often-encountered scenarios on contemporary commodity platforms.

Why bother?

Compiler is smart, one just needs to specify a correct command-line option, no?

```
for (int i = 0; i < m; i++) {
  for (int j = 0; j < n; j++) {
    for (int p = 0; p < k; p++) {
      C[i][j] += A[i][p] * B[p][j];
    }
  }
}
```

For vertical operations that is indeed not a complex task to do, and compilers perform just fine.

Why bother?

What if the level of triviality is reduced a little?

- Control flow dependencies.
- Different lane widths and parameters
- Branching
- Iterations of different length

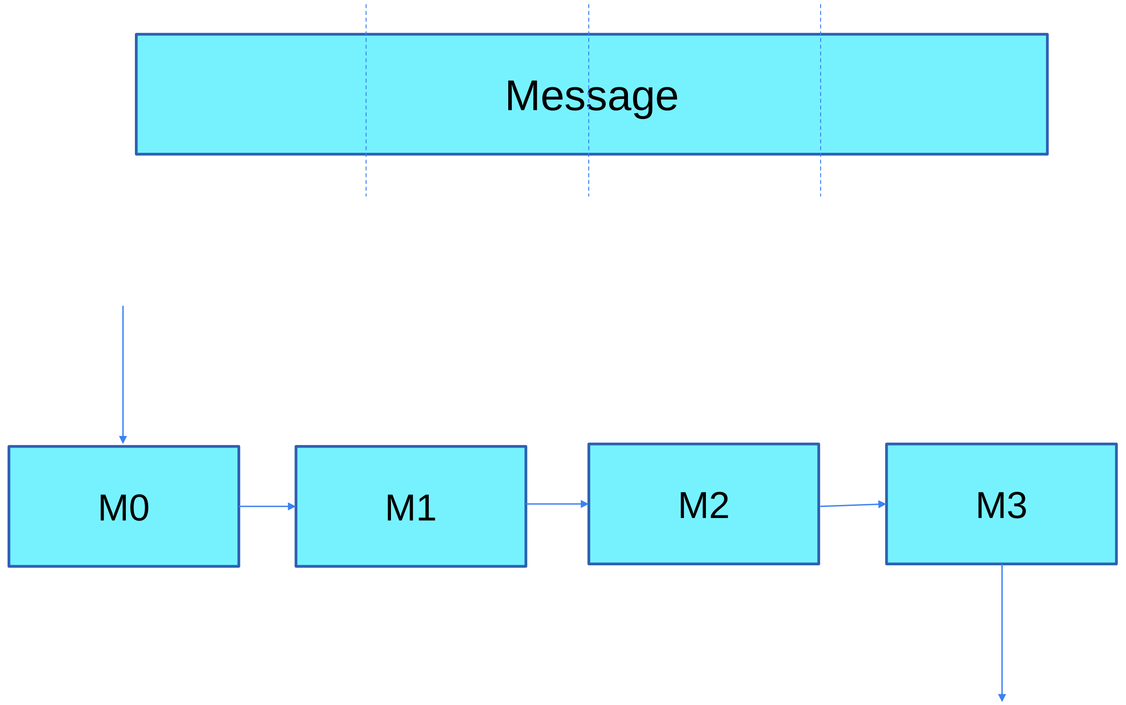
```
missed: not vectorized: complicated access pattern.
missed: not vectorized: no vectype for stmt: sum[0] = { 0, 0, 0, 0 };
missed: couldn't vectorize loop
```

```
uint32_t CityHash32(const char *s, size_t len) {

  if (len <= 24) {
    return len <= 12 ? (len <= 4 ?
     Hash32Len0to4(s, len) :
     Hash32Len5to12(s, len)) :
     Hash32Len13to24(s, len);
  }

  h ^= a0;
  h = h * 5 + 0xe6546b64;
  h ^= a2;

  do {
    h ^= a0;
    h = Rotate32(h, 18);
    h = h * 5 + 0xe6546b64;
    f += a1;
    s += 20;
  } while (--iters != 0);

  h = h * 5 + 0xe6546b64;
  h = Rotate32(h, 17) * c1;
  h = Rotate32(h + f, 19);
  h = h * 5 + 0xe6546b64;
  h = Rotate32(h, 17) * c1;

  return h;
}
```

SHA2 example

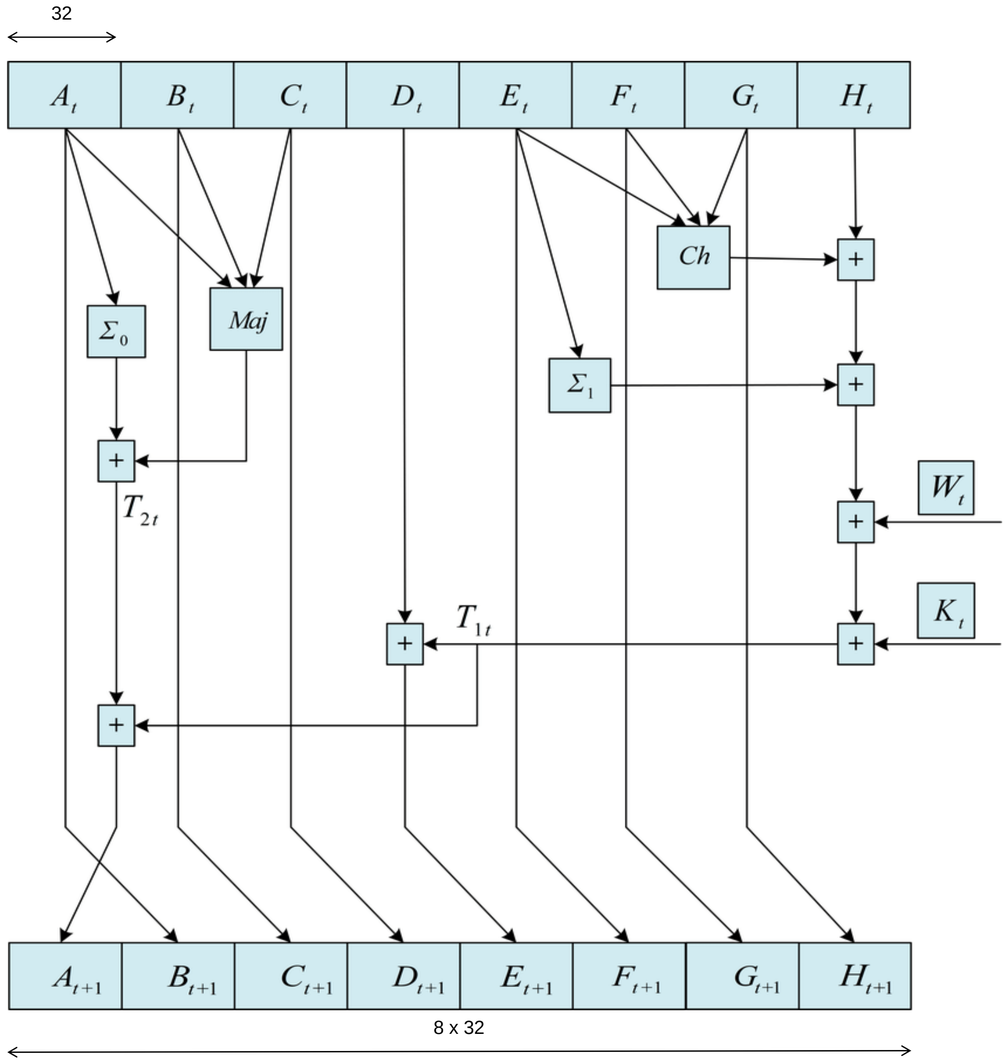
- A block-based cryptographic hash function, in widespread use.
- Initial message is split into fixed length blocks.
- The same transformation function is applied to each block.
- Transient state is carried between adjacent blocks only.

There are more tiny details of SHA2 that are not relevant to our discussion.



SHA2 example

- Each block is processed independently, no internal state is leaked outside.
- Simple bitwise operations – many of them.
- Each operation has a fixed width of 32 bits.
- It is a reasonably widespread operation, and processor vendors started providing specific instructions tailored specifically at SHA2 processing.
- Instruction set equivalence across different processor families may not be assumed.



SHA2 example – some results

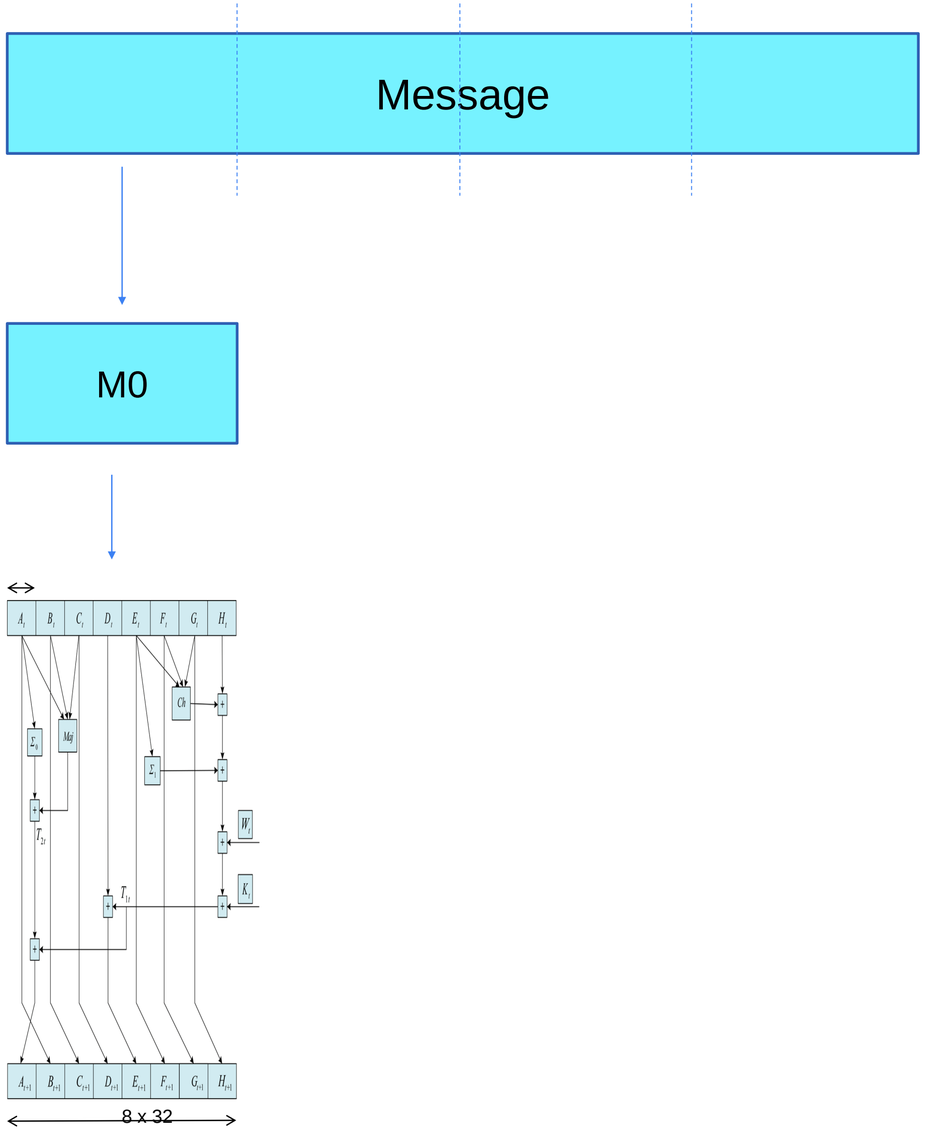
- Scalar implementation taken as a reference.
- Out of order and pipelining aspects considered equally for all methods.

- Microbenchmarking may not provide reliable system-level results!

	Latency	Throughput
Scalar	1	1
Accelerated	4	2
Vectorized	0.85	16

SHA2 experiments

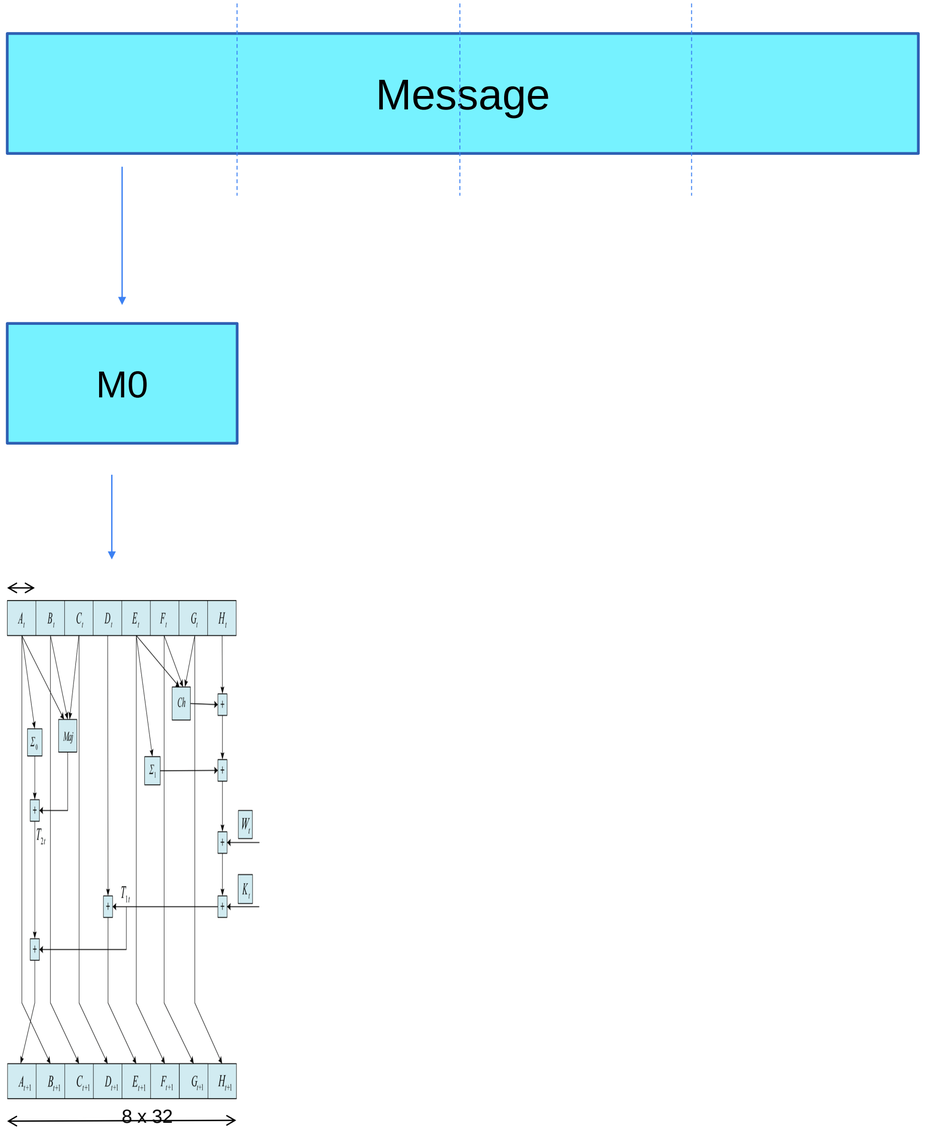
- Load full width register and perform all the operations of the SHA2 round on it.
- Just like a wider scalar version.



SHA2 experiments

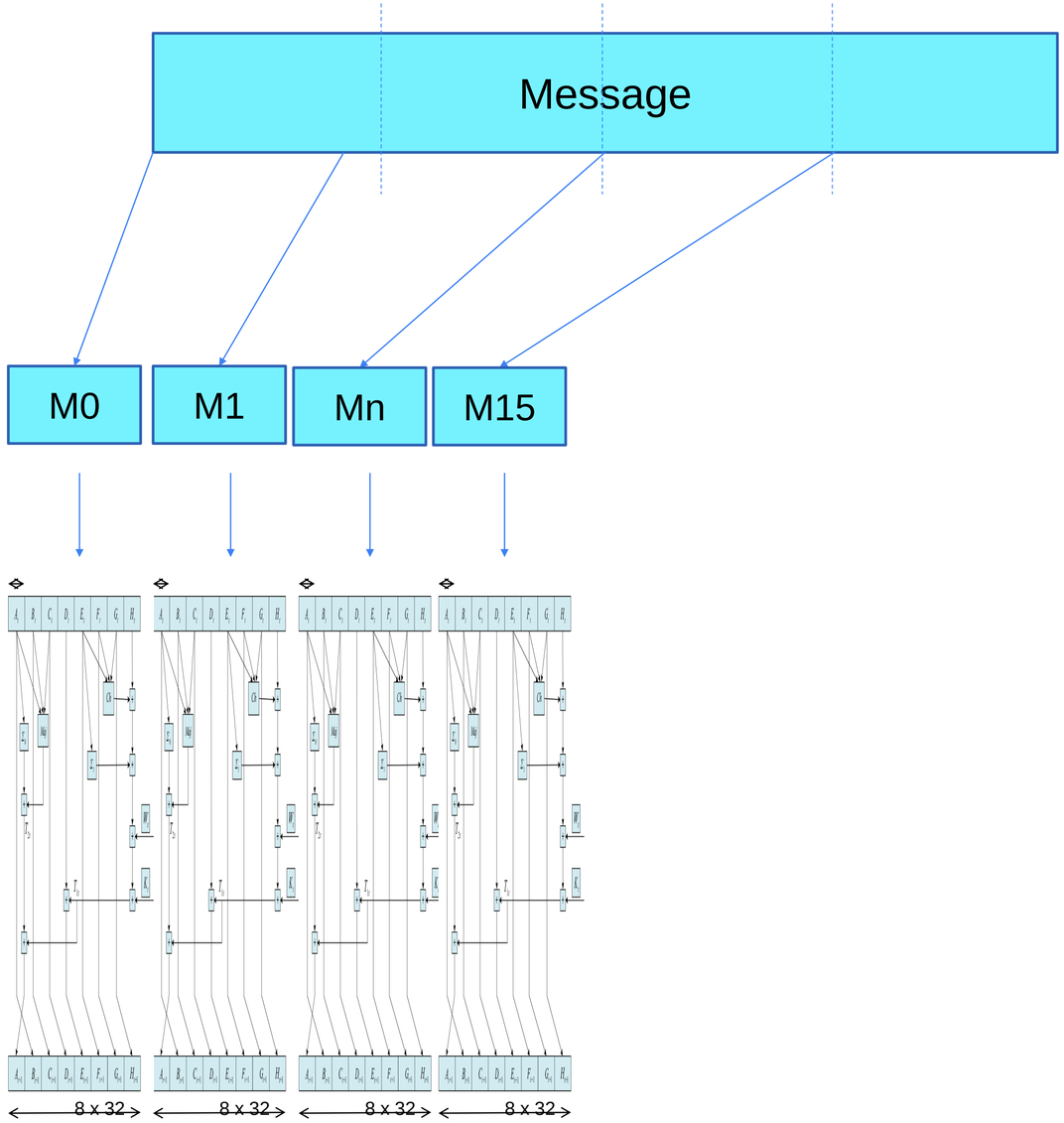
FAIL

- It does not work this way – most of vector operations are vertical, not horizontal.
- Wide memory operations hide memory hierarchy latency and are strongly preferred overall.
- Efficiency of memory subsystem operations is a byproduct of vectorization.



SHA2 experiments

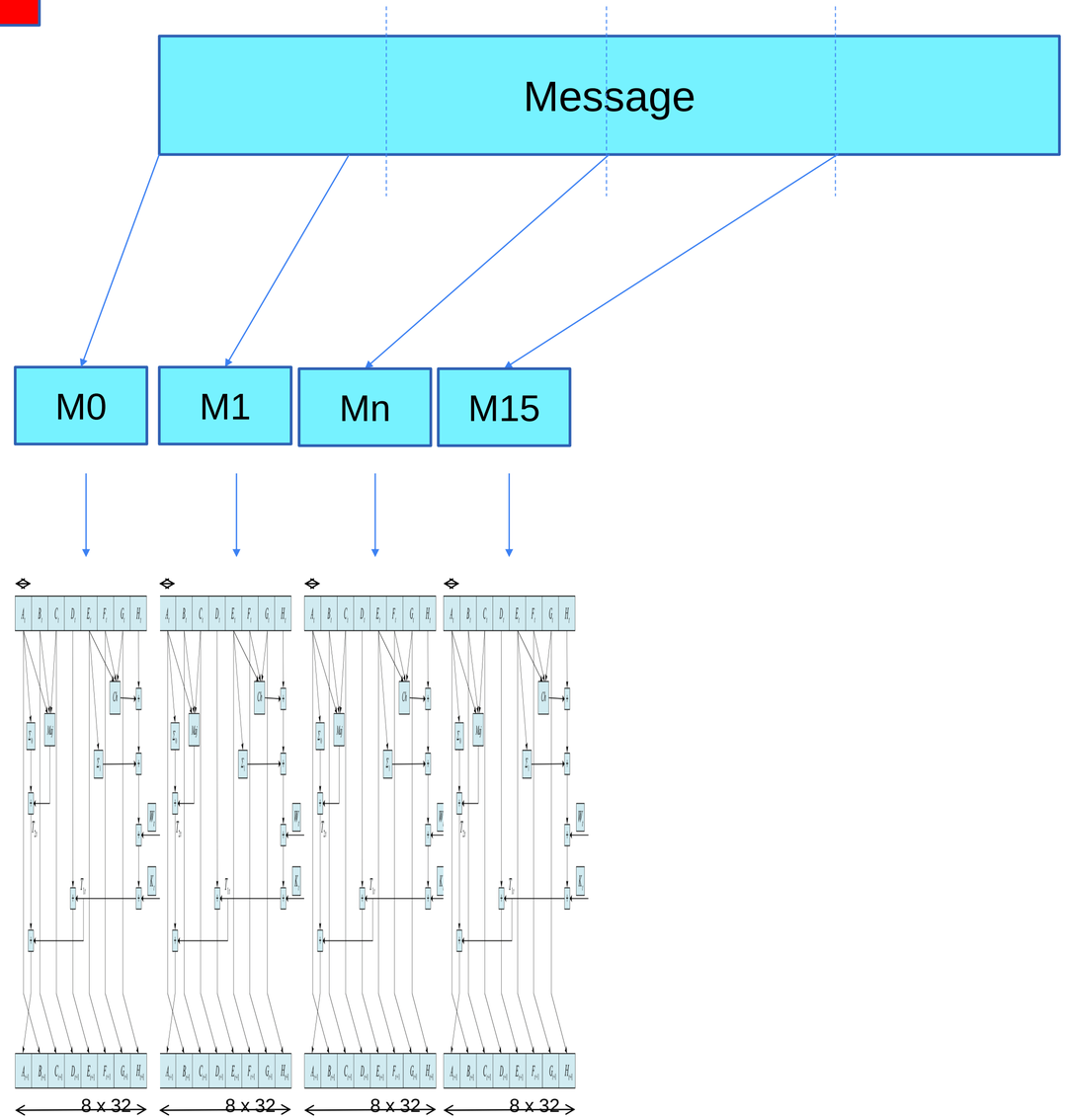
- Load multiple full registers.
- Transpose to vertical data layout.
- Perform a round on a set of registers



SHA2 experiments

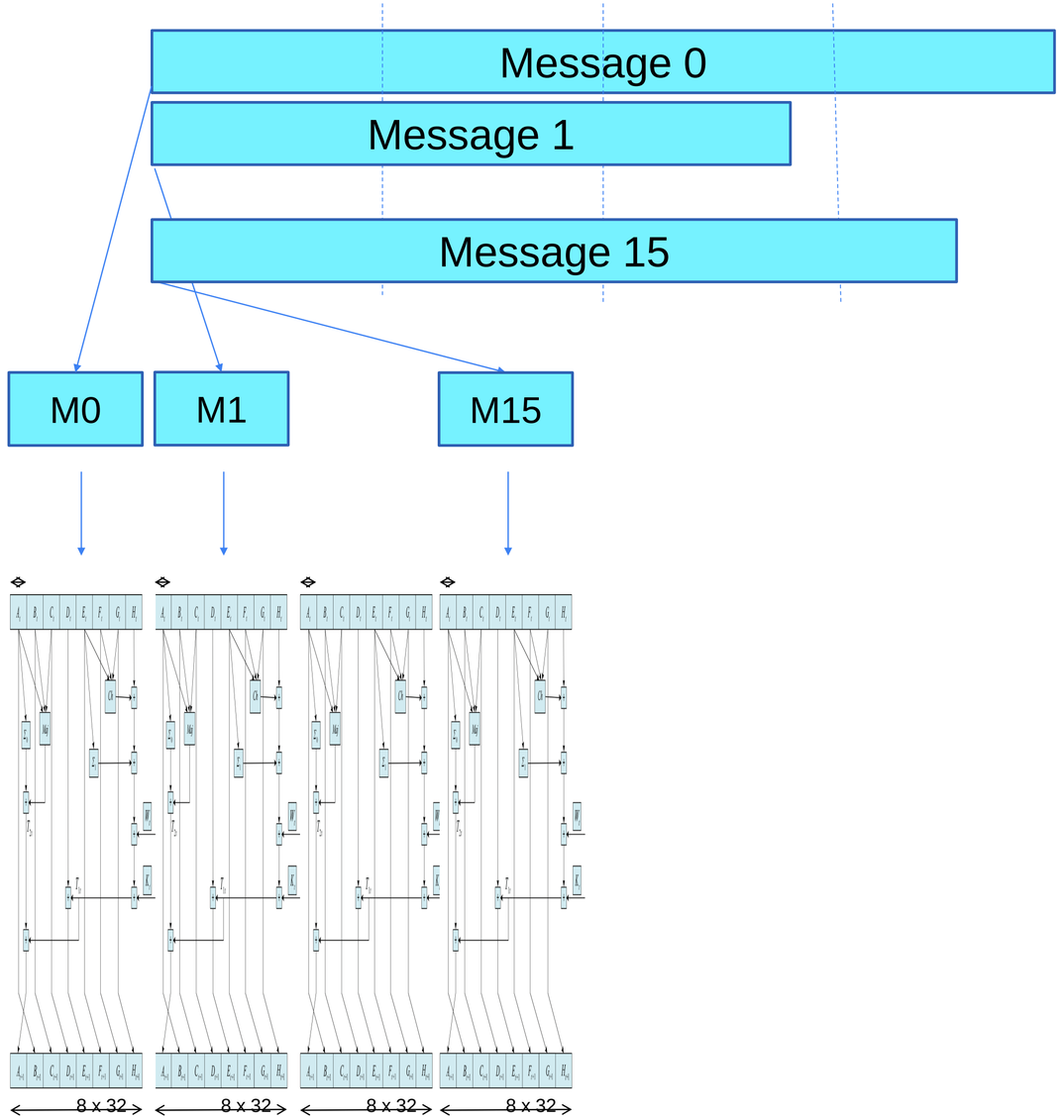
FAIL

- Blocks are bound together by an inter-block state transfer.
- Technically can be made to work, but at what cost?
- Resulting throughput -> 1, resulting latency -> +inf.



SHA2 experiments

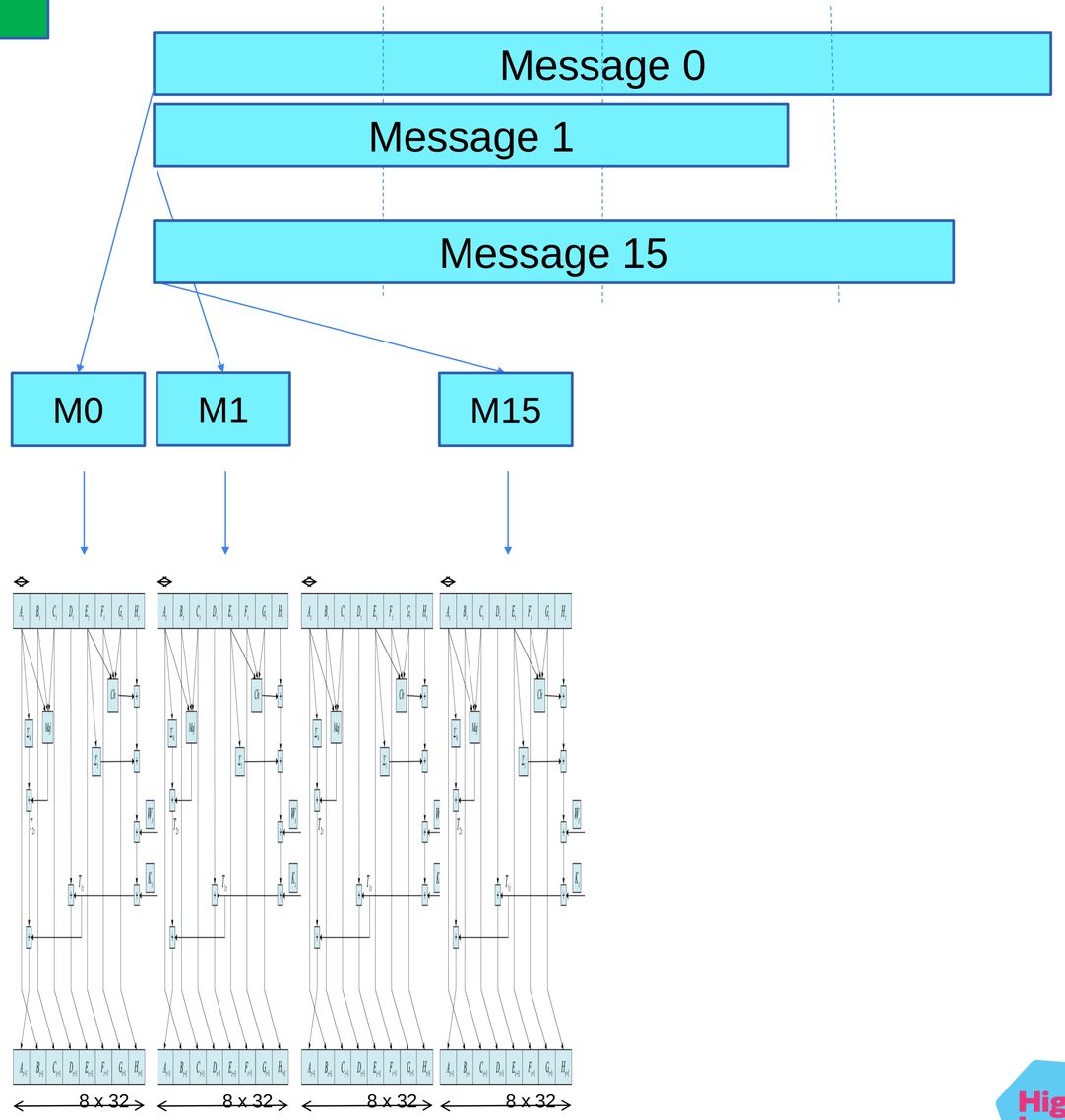
- Load multiple full registers with fragments of multiple messages.
- Transpose to vertical data layout.
- Perform a round on a set of registers



SHA2 experiments

PASS

- No cross-message block dependency.
- Throughput approaches vector width and vector element size ratio.
- Latency will be similar to a scalar option, many influencing factors.
- Latency might be better than scalar option in some specific cases – yet more influencing factors.

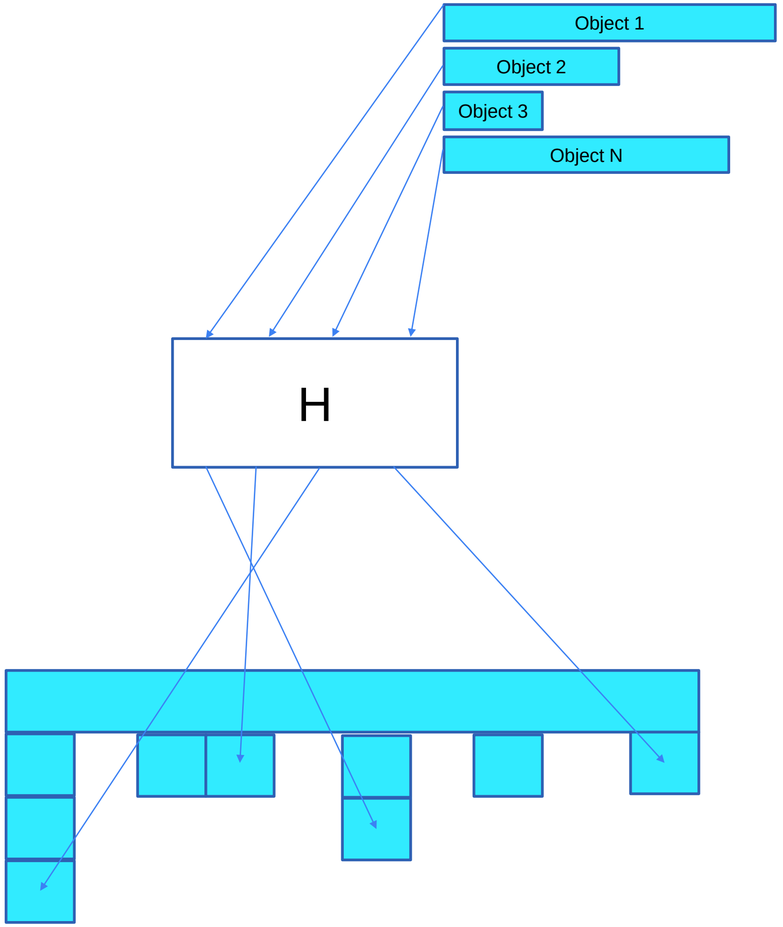


SHA2 experiments – some extrapolation

- Can this be autovectorized?
- Memory and caching subsystem influence.
- Handling of non-regular cases – both data flow and control flow.
- Specialty accelerators vs vectorization vs pipelined scalar execution.

Hash tables (maps)

- Two logical parts – calculate a hash value and place into corresponding storage.
- Memory intensive – but with different properties of intensity.
- Separate independent iterations.
- The hash function itself is comprised of arithmetical and bit operations.
- Logical and bit operations ideally fit vector element width.
- Multiplication can overflow though!
- Data fetch via gather or linear load plus transposition.
- Scalar prologue and epilogue might result in better overall performance.
- Linear execution time across all lanes, throughput vs goodput.



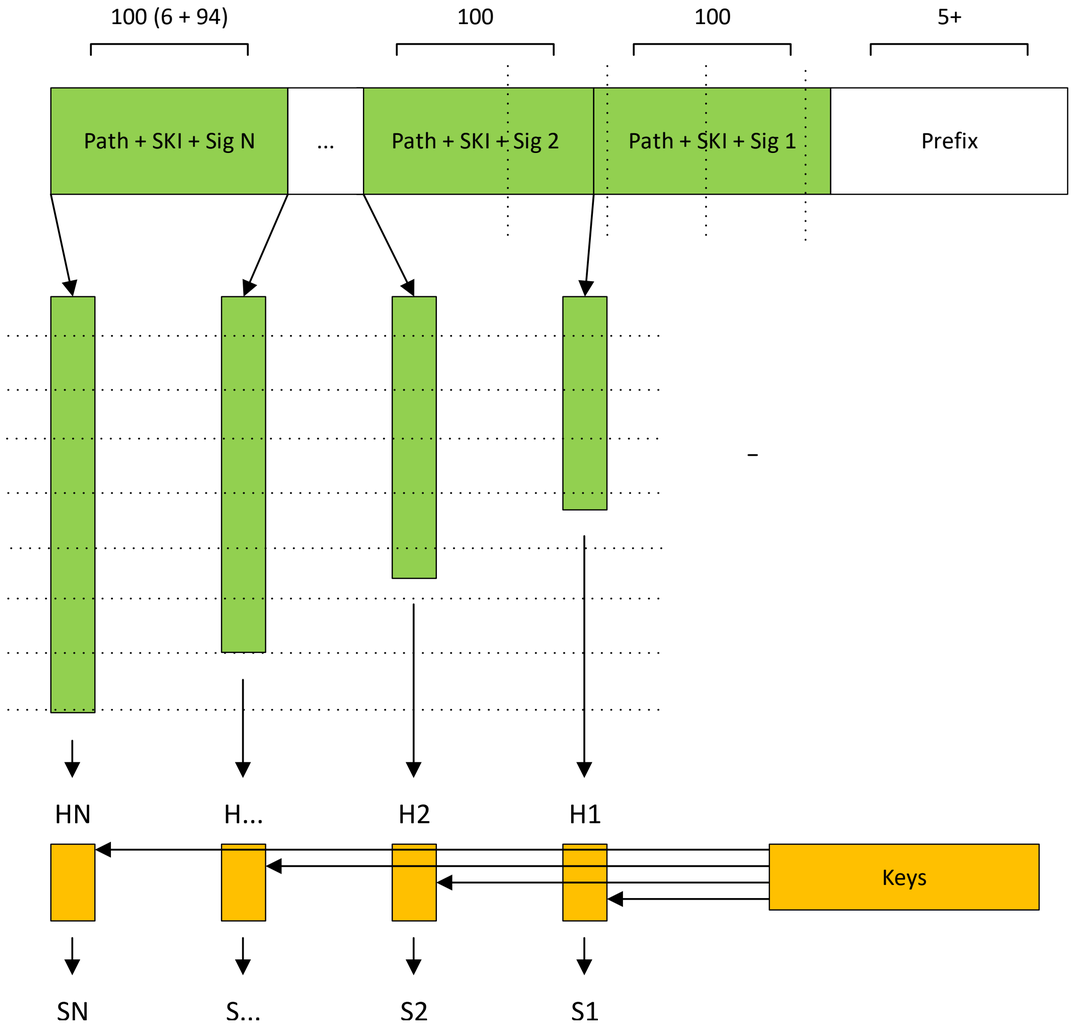
Tree-like Structures

- Tree traversal results in control flow branching. That needs to be translated into data flow masking.
- Initial pre-sorting based on search criteria parameters may result in observable performance increase.
- Dealing with 64-bit pointers when only 32-bit elements are available results in substantial slowdown.
- Unrolled scalar pass may result in comparable or better performance overall.

Long Integer arithmetic

- Widely used in cryptography.
- A large integer is represented as a set of narrow limbs.
- Operations performed on limbs vertically.
- Specific algorithms generally stay the same as for a scalar approach.
- Cost of arithmetic operations generally outweighs memory access.
- “Free” constant-time execution for a group of lanes.

Combining it all together- BGPsec



Linear code block operating on different data sets in parallel

Hash multiple blocks in parallel
Sign/verify multiple hashes/signatures in parallel

Vector lanes of fixed width – multiplication restricts lane density

Gather operations place significant restrictions on data format

+20% latency results in +1200% throughput

Only if data structures allow!

Where’s the complexity anyway?

- In order to vectorize scalar code successfully, flexible horizontal operations are mandatory.
- Control flow dependencies need to be translated to data flow dependencies.
- Data structure layout should not conflict with native vector width and length.

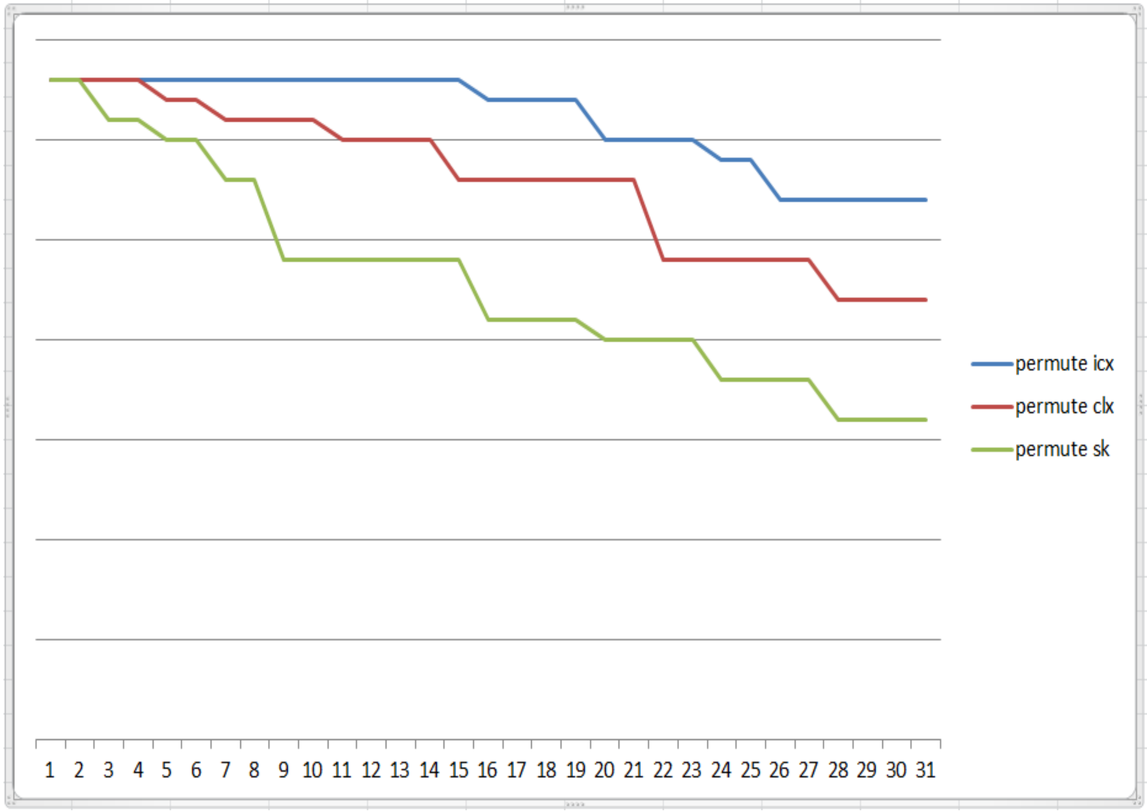
  - Controllable/maskable access to memory
  - Assembling a vector register from scalar values (insert/extract)
  - Mixing multiple vectors into one (blend)
  - Swapping blocks and elements within a vector (permute, shuffle)
  - Nonlinear and indexed memory access (scatter, gather)
  - Conditional execution (predicates, masking)

Assorted Aspects

- Atomicity of vector operations.
- Memory ordering.
- Domain-specific extensions (FMA52, GF2, AMX).
- Interworking with an OS, state management and context switching.
- Alignment, speculative execution and prefetching.

Controversial Topics

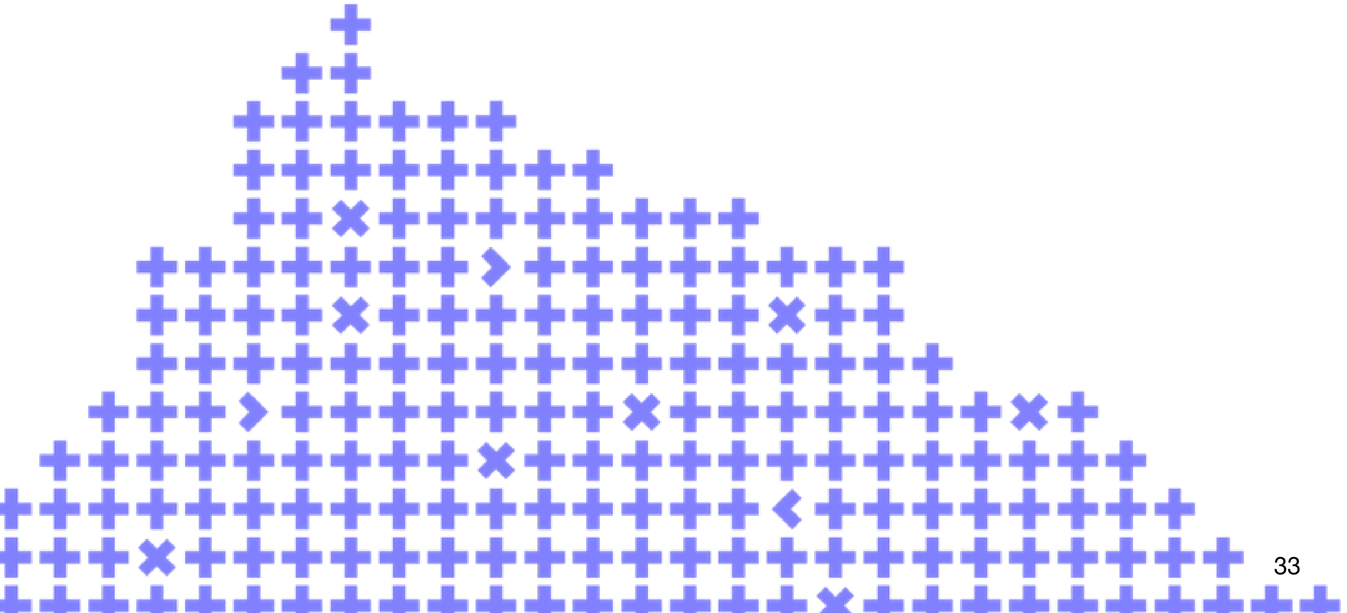
- Intrinsics in 2022?
- Is AVX-512 obsolete?
- What about alternative platforms?
  ARM, RISC-V?
- What about portability?
- What about compiler support?
- What's next beyond AVX-512?
- Variable vector length and width?



Discussion

Leave your feedback!

You can rate the talk and give a feedback on what you've liked or what could be improved



Co-organizer

Yandex